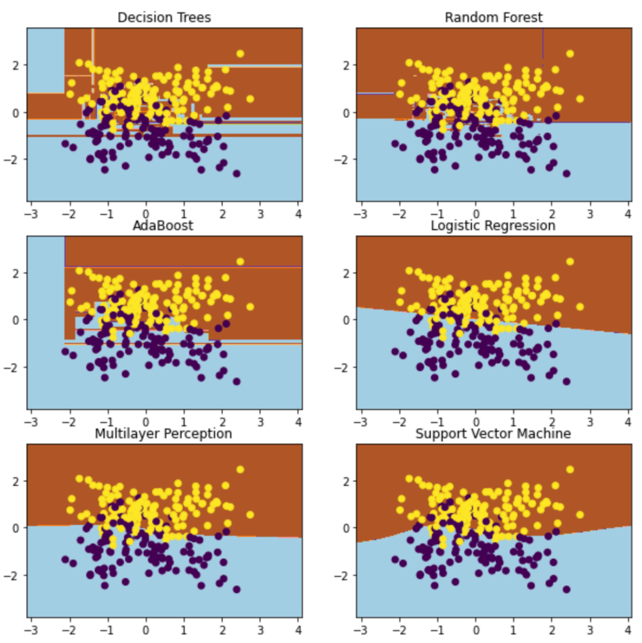
Question 1

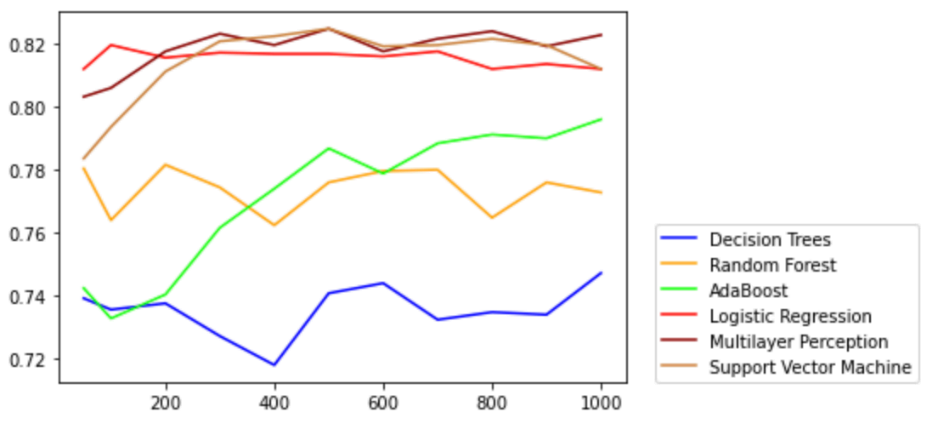
(a)



```python
def create_dataset(n=1250, nf=2, nr=0, ni=2, random_state=125):
    '''
    generate a new dataset with
    n: total number of samples
    nf: number of features
    nr: number of redundant features (these are linear combinatins of informative features)
    ni: number of informative features (ni + nr = nf must hold)
    random_state: set for reproducibility
    '''
    X, y = make_classification( n_samples=n,
                                n_features=nf,
                                n_redundant=nr,
                                n_informative=ni,
                                random_state=random_state,
                                n_clusters_per_class=2)
    rng = np.random.RandomState(2)
    X += 3 * rng.uniform(size = X.shape)
    X = StandardScaler().fit_transform(X)
    return X, y

def plotter(classifier, X, X_test, y_test, title, ax):
    # plot decision boundary for given classifier
    plot_step = 0.02
    x_min, x_max = X[:, 0].min() - 1, X[:,0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:,1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                         np.arange(y_min, y_max, plot_step))
    Z = classifier.predict(np.c_[xx.ravel(),yy.ravel()])
    Z = Z.reshape(xx.shape)
    if ax:
        ax.contourf(xx, yy, Z, cmap = plt.cm.Paired)
        ax.scatter(X_test[:, 0], X_test[:, 1], c = y_test)
        ax.set_title(title)
    else:
        plt.contourf(xx, yy, Z, cmap = plt.cm.Paired)
        plt.scatter(X_test[:, 0], X_test[:, 1], c = y_test)
        plt.title(title)
```

```python
#Question1 (a)
X, y = (create_dataset(n=1250, nf=2, nr=0, ni=2, random_state=125))
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=45)
clf0 = DecisionTreeClassifier()
clf1 = RandomForestClassifier()
clf2 = AdaBoostClassifier()
clf3 = LogisticRegression()
clf4 = MLPClassifier()
clf5 = SVC()

clfs = [clf0, clf1, clf2, clf3, clf4, clf5]
title = ['Decision Trees', 'Random Forest', 'AdaBoost', 'Logistic Regression', 'Multilayer Perception',
         'Support Vector Machine']
fig, ax = plt.subplots(3,2, figsize=(10,10))

for i in range(len(clfs)):
    clfs[i].fit(X_train, y_train)
    plotter(clfs[i], X, X_test, y_test, title[i], ax[i//2,i%2])  #i//2:row i%2: col
```

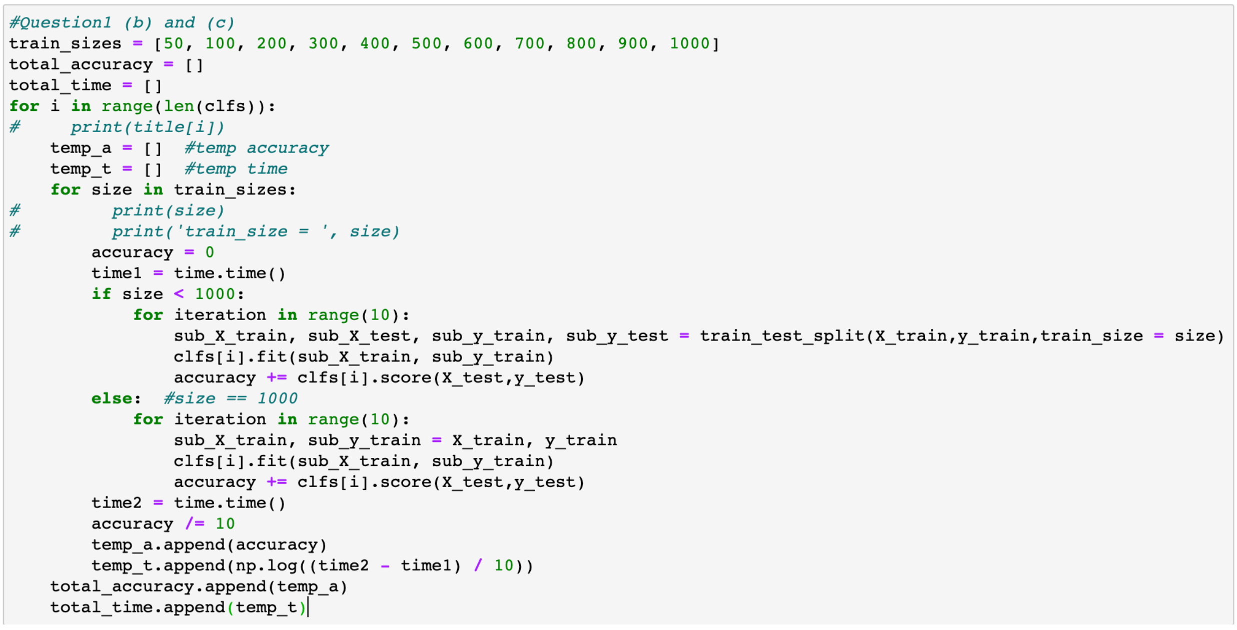
(b)



According to the picture above, it could be concluded that the average accuracy of Multilayer Perception, Support Vector Machine and Logistic Regression is greater than other models with all different train sizes. In addition, Decision Tree performs the worst in this specific task. Recall that Bias is the difference between the average prediction of our model and the correct value which we are trying to predict (week1 lecture notes), it is clear that the Multilayer Perception, SVM and Logistic Regression have lower bias, while the Decision Tree has the highest bias.

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data (week1 lecture notes). The accuracy on the test set of AdaBoost, Random Forest and Decision Tree have more remarkable fluctuation on different train size, which means they have greater variance and cannot generalize well on the test data. This explains why their accuracy tends to be lower.

Balance the bias and variance, I prefer the Multilayer Perception for this task, which has lower bias and lower variance, as well as performs the best with train size set to 1000.

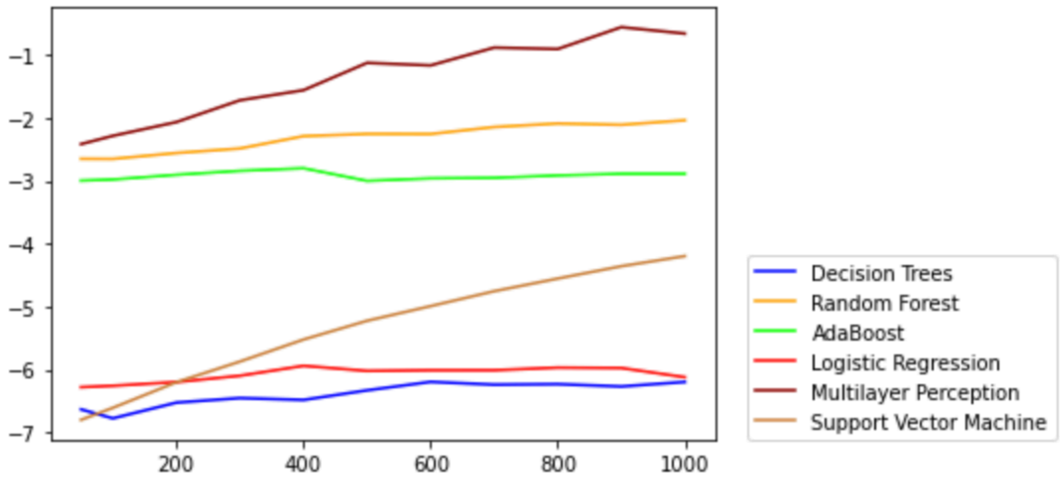


```
#Question1 (b) and (c)
train_sizes = [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
total_accuracy = []
total_time = []
for i in range(len(clfs)):
#     print(title[i])
    temp_a = []  #temp accuracy
    temp_t = []  #temp time
    for size in train_sizes:
#         print(size)
#         print('train_size = ', size)
        accuracy = 0
        time1 = time.time()
        if size < 1000:
            for iteration in range(10):
                sub_X_train, sub_X_test, sub_y_train, sub_y_test = train_test_split(X_train,y_train,train_size = size)
                clfs[i].fit(sub_X_train, sub_y_train)
                accuracy += clfs[i].score(X_test,y_test)
        else:  #size == 1000
            for iteration in range(10):
                sub_X_train, sub_y_train = X_train, y_train
                clfs[i].fit(sub_X_train, sub_y_train)
                accuracy += clfs[i].score(X_test,y_test)
        time2 = time.time()
        accuracy /= 10
        temp_a.append(accuracy)
        temp_t.append(np.log((time2 - time1) / 10))
    total_accuracy.append(temp_a)
    total_time.append(temp_t)
```

```
colors = ['blue', 'orange', 'lime', 'red' , 'darkred', 'peru']

for i in range(len(total_accuracy)):
    plt.plot(train_sizes, total_accuracy[i], color = colors[i], label = title[i])
    plt.legend(bbox_to_anchor=(1.05, 0), loc=3, borderaxespad=0)
    ### reference: https://blog.csdn.net/Poul_henry/article/details/82533569
plt.show()
```

Parameters for setting the location of the legends, reference : https://blog.csdn.net/Poul_henry/article/details/82533569

(C)



Based on above picture, the train time for different models:
Multilayer Perception > Random Forest > AdaBoost > SVM > Logistic Regression > DT.
This is related to the characters of the models. The more complex the model is, the more parameters it has, and the more time would be required for calculation and updating the parameters.
In general, the train time grows as the train size increases. However, the train time for Random Forest, AdaBoost, Logistic Regression and DT grows much more slightly than Multilayer Perception and SVM. This means that SVM and Multilayer Perception are more sensitive to the train size.

```
for i in range(len(total_time)):
    plt.plot(train_sizes, total_time[i], color = colors[i], label = title[i])
    plt.legend(bbox_to_anchor=(1.05, 0), loc=3, borderaxespad=0)
plt.show()
```

A part of codes is same as that in 1.(b).

(d)

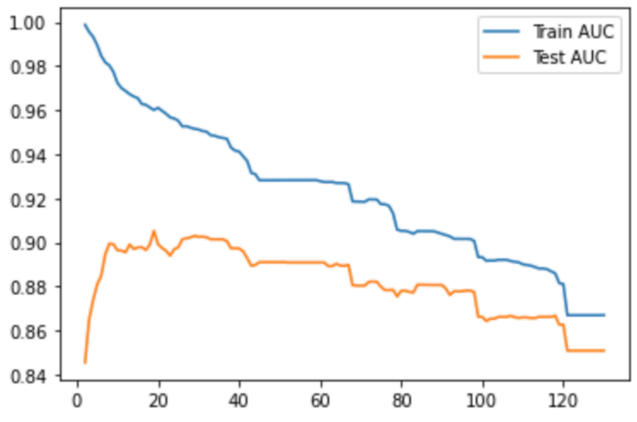
Train accuracy: 1.0

Test accuracy: 0.814

```
X, y = create_dataset(n=2000, nf = 20, nr = 12, ni = 8, random_state = 25)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.5,random_state=15)
DT_clf = DecisionTreeClassifier(random_state = 0)
DT_clf.fit(X_train, y_train)
train_ac = DT_clf.score(X_train,y_train)
test_ac = DT_clf.score(X_test, y_test)
print('Accuracy on the train set: ', train_ac)
print('Accuracy on the test set: ', test_ac)
```

```
Accuracy on the train set:  1.0
Accuracy on the test set:  0.814
```

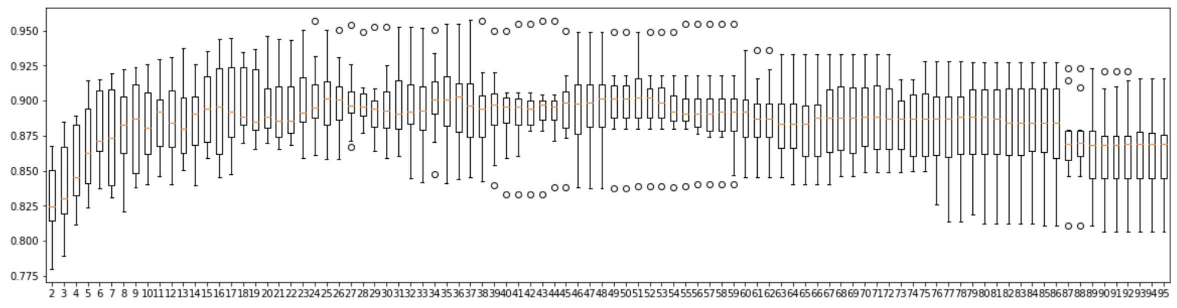
(e)



```
import sklearn

leafs = [i for i in range(2, 131)]
train_AUC = []
test_AUC = []
for leaf in leafs:
    DT_clf = DecisionTreeClassifier(min_samples_leaf = leaf, random_state = 0)
    DT_clf.fit(X_train, y_train)
    train_predict = DT_clf.predict_proba(X_train)[:,1]
    trainauc = sklearn.metrics.roc_auc_score(y_train,train_predict)

    test_predict = DT_clf.predict_proba(X_test)[:,1]
    testauc = sklearn.metrics.roc_auc_score(y_test, test_predict)
    train_AUC.append(trainauc)
    test_AUC.append(testauc)
plt.plot(leafs, train_AUC,label = 'Train AUC')
plt.plot(leafs, test_AUC, label = 'Test AUC')
plt.legend()
plt.show()
```

(f)

```
The max score is  0.9025196654853911 , with k =  24
```



The highest CV score is 0.9025, with k = 24.
DT with min_samples_leaf = 24,
Train accuracy: 0.881
Test accuracy: 0.825

```python
leafs = [i for i in range(2, 96)]
folds = 10
total_auc = []
max_score = 0
max_k = -1
for leaf in leafs:
    temp_auc = []
    current_auc = 0
    for i in range(folds):
        DT_clf = DecisionTreeClassifier(min_samples_leaf = leaf, random_state = 0)
        begin = int((len(X_train) / folds) * i)
        end = int((len(X_train) / folds) * (i + 1))
        cv_X = X_train[begin:end]
        cv_y = y_train[begin:end]
        other_X = np.vstack((X_train[0:begin],X_train[end:]))
        other_y = np.hstack((y_train[0:begin],y_train[end:]))
        DT_clf.fit(other_X, other_y)
        cv_predict = DT_clf.predict_proba(cv_X)[:,1]
        cvauc = sklearn.metrics.roc_auc_score(cv_y, cv_predict)
        temp_auc.append(cvauc)
        current_auc += cvauc
    total_auc.append(temp_auc)
    if current_auc / 10 > max_score:
        max_score = current_auc / 10
        max_k = leaf

print('The max score is ', max_score, ', with k = ', max_k)
plt.figure(figsize = (20,5))
plt.tight_layout()
plt.boxplot(total_auc, labels = leafs)
plt.show()
```

```
DT_clf = DecisionTreeClassifier(min_samples_leaf = 24, random_state = 0)
DT_clf.fit(X_train, y_train)
train_ac = DT_clf.score(X_train,y_train)
test_ac = DT_clf.score(X_test, y_test)
print(train_ac, test_ac)
```

```
0.881 0.825
```

(g)

Optimal min_samples_leaf: 28
Train accuracy: 0.952045204520452
Test accuracy: 0.9022368384203304

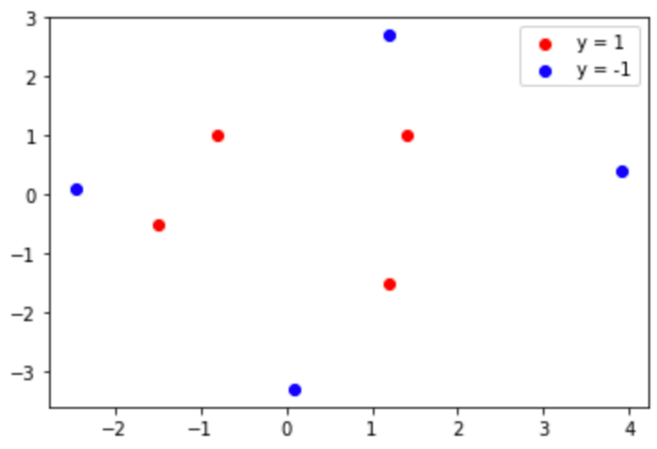
```
from sklearn.model_selection import GridSearchCV
#leafs = [i for i in range(2, 96)]
grid = GridSearchCV(DecisionTreeClassifier(random_state = 0),param_grid={'min_samples_leaf': leafs},cv=10,
                   scoring = 'roc_auc')
grid.fit(X_train, y_train)
print(grid.best_params_)
train_ac = grid.score(X_train,y_train)
test_ac = grid.score(X_test, y_test)
print(train_ac, test_ac)
```

```
{'min_samples_leaf': 28}
0.952045204520452 0.9022368384203304
```

By GridSearchCV, the optimal min_samples_leaf is 28, which is different from that in (e) (min_samples_leaf = 24). This is because that the method used in GridSearchCV is stratified K fold, which is different from K fold CV. In GridSearchCV, the proportion of the samples with different labels in the test set is same with that in the train set, while K fold CV (in (e)) does not hold this character.

Question 2

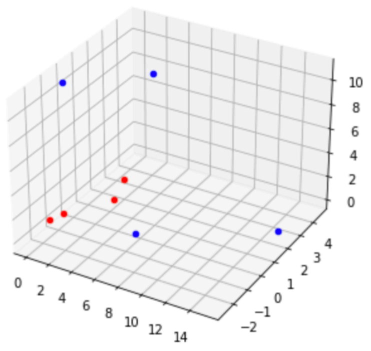
(a)



Set m = 0 , d = 2, (based on the plot,) it could be concluded that the data becomes linearly separable.

```
##question 2 (a)
X = np.array([[-0.8, 1], [3.9, 0.4], [1.4,1], [0.1,-3.3], [1.2,2.7],[-2.45,0.1], [-1.5,-0.5],
             [1.2,-1.5]])
y = np.array([1,-1,1,-1,-1,-1,1,1])
color = {-1: 'b', 1:'r'}
for i in range(len(X)):
    plt.scatter(X[i][0], X[i][1], c = color[y[i]])
plt.legend(('y = 1', 'y = -1'))
plt.show()
```

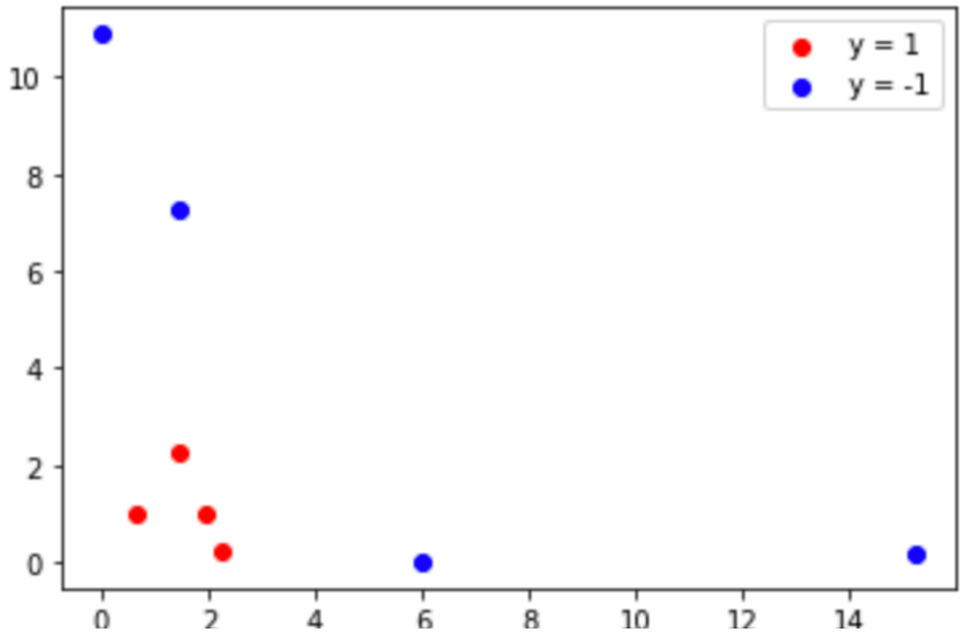
```
#if set m = 0, d = 1, k(x,y) = (x.T * y)
#based on 2.1, the data is linearly unseparable
#then set m = 0, d = 2, k(x,y) = (x.T * y) ** 2
#the 3d representation:
new = []
for i in range(len(X)):
    new.append([X[i][0] ** 2, np.sqrt(2) * X[i][0] * X[i][1], X[i][1] ** 2])
new = np.array(new)

# from mpl_toolkits.mplot3d import Axes3D
# fig = plt.figure()
# ax = Axes3D(fig)
# for i in range(len(new)):
#     ax.scatter(new[i][0], new[i][1], new[i][2], c = color[y[i]])
```



(b)

Feature Representation:

```
[[ 6.40000000e-01 -1.13137085e+00  1.00000000e+00]
 [ 1.52100000e+01  2.20617316e+00  1.60000000e-01]
 [ 1.96000000e+00  1.97989899e+00  1.00000000e+00]
 [ 1.00000000e-02 -4.66690476e-01  1.08900000e+01]
 [ 1.44000000e+00  4.58205194e+00  7.29000000e+00]
 [ 6.00250000e+00 -3.46482323e-01  1.00000000e-02]
 [ 2.25000000e+00  1.06066017e+00  2.50000000e-01]
 [ 1.44000000e+00 -2.54558441e+00  2.25000000e+00]]
```

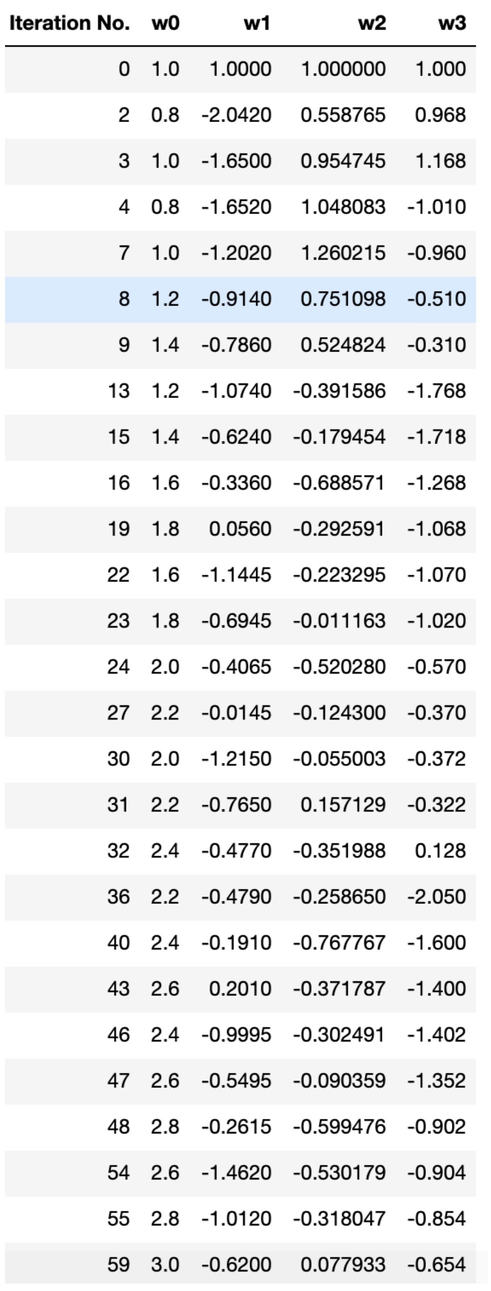


Based on the picture above, the data is linearly separable.

```
print(new)
a = new[:,0]
b = new[:,1]
c = new[:,2]

#choose a and c, i.e. the 1st and the 3rd in the 3-d vector 'new'
d = np.array([a,c]).T
for i in range(len(X)):
    plt.scatter(d[i][0], d[i][1], c = color[y[i]])
plt.legend(('y = 1', 'y = -1'))
plt.show()
```

(c)

Table of iterations:



Iteration No.	w0	w1	w2	w3
0	1.0	1.0000	1.000000	1.000
2	0.8	-2.0420	0.558765	0.968
3	1.0	-1.6500	0.954745	1.168
4	0.8	-1.6520	1.048083	-1.010
7	1.0	-1.2020	1.260215	-0.960
8	1.2	-0.9140	0.751098	-0.510
9	1.4	-0.7860	0.524824	-0.310
13	1.2	-1.0740	-0.391586	-1.768
15	1.4	-0.6240	-0.179454	-1.718
16	1.6	-0.3360	-0.688571	-1.268
19	1.8	0.0560	-0.292591	-1.068
22	1.6	-1.1445	-0.223295	-1.070
23	1.8	-0.6945	-0.011163	-1.020
24	2.0	-0.4065	-0.520280	-0.570
27	2.2	-0.0145	-0.124300	-0.370
30	2.0	-1.2150	-0.055003	-0.372
31	2.2	-0.7650	0.157129	-0.322
32	2.4	-0.4770	-0.351988	0.128
36	2.2	-0.4790	-0.258650	-2.050
40	2.4	-0.1910	-0.767767	-1.600
43	2.6	0.2010	-0.371787	-1.400
46	2.4	-0.9995	-0.302491	-1.402
47	2.6	-0.5495	-0.090359	-1.352
48	2.8	-0.2615	-0.599476	-0.902
54	2.6	-1.4620	-0.530179	-0.904
55	2.8	-1.0120	-0.318047	-0.854
59	3.0	-0.6200	0.077933	-0.654

Final weight vector:

$[\,3, -0.62, 0.07793276, \quad -0.654]$

x_i	$\phi\left(x_i\right)$	$y_i\phi^T\left(x_i\right)w*$
$(-0.8,1)^T$	$(0.64,-1.13137085,1.0)^T$	>0
$(3.9,0.4)^T$	$(15.21,2.20617316,0.16)^T$	>0
$(1.4,1)^T$	$(1.96,1.97989899,1.0)^T$	>0
$(0.1,-3.3)^T$	$(0.01,-0.466690476,10.89)^T$	>0
$(1.2,2.7)^T$	$(1.44,4.58205194,7.29)^T$	>0
$(-2.45,0.1)^T$	$(6.0025,-0.34648232,0.01)^T$	>0
$(-1.5,-0.5)^T$	$(2.25,1.06066017,0.25)^T$	>0
$(1.2,-1.5)^T$	$(1.44,-2.54558441,2.25)^T$	>0

```
###(c)
w = np.ones(4)
new_X = np.insert(new, 0, values = np.ones(len(new)), axis = 1)  #insert a col with values = 1 to X
learning_rate = 0.2
#print(new_X)
flag = 0

total_w = []
total_w.append(w)     #initialized with [(1,1,1,1)]
iteration_list = []
iteration_list.append(0)     #initialized with [0]
iteration = 0
while(flag == 0):
    flag = 1
    for i in range(len(new_X)):
        value = np.dot(w, new_X[i])
        iteration += 1
        if value * y[i] > 0:         #if classified rightly
            continue
        else:          #misclassified
            flag = 0
            w = w + learning_rate * y[i] * new_X[i]
            print('iteration = ', iteration, '\nw = ', w)
            iteration_list.append(iteration)
            total_w.append(w)
print('Final weight vector: ', w)
```

```
#print the iteration table
import pandas as pd
df = pd.DataFrame(columns=('Iteration No.', 'w0', 'w1', 'w2', 'w3'))
df.head()
iteration = 0
i = 0
for item in total_w:
    df.loc[i] = {'Iteration No.': iteration_list[i] ,'w0':item[0], 'w1':item[1], 'w2':item[2], 'w3':item[3]}
    iteration += 1
    i += 1
df.head(30)
```

```
#if the perceptron has converged
for i in range(len(X)):
    print(X[i])
    print(new[i])
    value = np.dot(new_X[i], w) * y[i]
    print(value > 0)
```

(d)

$n = 2, k = (1 + x_1y_1)(1 + x_2y_2) = 1 + x_1y_1 + x_2y_2 + x_1x_2y_1y_2$

$Feature\ Representation : \langle (1, x_1, x_2, x_1x_2),\ \ (1, y_1, y_2, y_1y_2) \rangle$

$$\begin{aligned} n = 3, k &= (1 + x_1y_1)(1 + x_2y_2)(1 + x_3y_3) \\ &= 1 + x_1y_1 + x_2y_2 + x_3y_3 + x_1x_2y_1y_2 + x_1x_3y_1y_3 + x_2x_3y_2y_3 + x_1x_2x_3y_1y_2y_3 \\ &= \Big\langle \big(1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1x_2x_3\big),\ \ \big(1, y_1, y_2, y_3, y_1y_2, y_1y_3, y_2y_3, y_1y_2y_3\big) \Big\rangle \end{aligned}$$

Thus, k(x,y) is the dot product of two vectors, one of which consists of xi and one consists of yi. The two vectors have similar formats, i.e. the first item is 1, and the rest items are equal with selecting 1, 2, 3, …, n items from (x1,x2,x3,…,xn) (or from (y1,y2,y3,…,yn)).
It could be written as <1, function(1), function(2), function(3), …., function(n) >, where function() is defined as:

```
$def\ function(n):$
    $a = select\ n\ different\ items\ from\ a\ set(x\ or\ y)\ regardless\ of\ the\ order$
    $return\ a$
```

Regardless of the order means we consider x1x2 and x2x1 the same, x1x2x3 ,x2x1x3, x1x3x2, x2x1x3, … the same, etc.
For example, n=4, k=<(1, function(1), function(2), function(3), function(4)(on x)),
(1, function(1), function(2), function(3), function(4)(on y)>
Where function(1) = x1, x2, x3, x4 (or y)
function(2) = x1x2, x1x3, x1x4, x2x3, x2x4, x3x4 (or y)
function(3) = x1x2x3, x1x2x4, x1x3x4, x2x3x4 (or y)
function(4) = x1x2x3x4 (or y)

$$\begin{aligned} i.e.\ n = 4, k = <\ &\big(x_1, x_2, x_3, x_4, x_1x_2, x_1x_3, x_1x_4, x_2x_3, x_2x_4, x_3x_4, x_1x_2x_3, x_1x_2x_4, x_1x_3x_4, x_2x_3x_4, x_1x_2x_3x_4\big), \\ &\big(y_1, y_2, y_3, y_4, y_1y_2, y_1y_3, y_1y_4, y_2y_3, y_2y_4, y_3y_4, y_1y_2y_3, y_1y_2y_4, y_1y_3y_4, y_2y_3y_4, y_1y_2y_3y_4\big) > \end{aligned}$$

$$\begin{aligned} n = n, k = <\ &\Big(x_1, x_2, \ldots, x_n, x_1x_2, .., x_1x_n, \ldots, x_{(n-1)}x_n, x_1x_2x_3, \ldots, x_{(n-2)}x_{(n-1)}x_n, \ldots, x_1x_2\ldots x_n\Big), \\ &\Big(y_1, y_2, \ldots, y_n, y_1y_2, \ldots, y_1y_n, \ldots, y_{(n-1)}y_n, y_1y_2y_3, \ldots, y_{(n-2)}y_{(n-1)}y_n, \ldots, y_1y_2\ldots y_n\Big) > \end{aligned}$$